8-21-00

A

Please type a plus sign (+) inside this box → [+]

PTO/SB/05 (4/98)
Approved for use through 09/30/2000. OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 C.F.R. § 1.53(b))

Attorney Docket No.	103.1035.01
First Inventor or Application Identifier	Lewis, Blake
Title	Instant Snapshot
Express Mail Label No.	EL 524 780 239 US

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO: Assistant Commissioner for Patents
Box Patent Application
Washington, DC 20231

1. [ ] * Fee Transmittal Form *(e.g., PTO/SB/17)*
(Submit an original and a duplicate for fee processing)

2. [X] Specification [Total Pages 36]
(preferred arrangement set forth below)
- Descriptive title of the Invention
- Cross References to Related Applications
- Statement Regarding Fed sponsored R & D
- Reference to Microfiche Appendix
- Background of the Invention
- Brief Summary of the Invention
- Brief Description of the Drawings *(if filed)*
- Detailed Description
- Claim(s)
- Abstract of the Disclosure

3. [X] Drawing(s) *(35 U.S.C. 113)* [Total Sheets 3]

4. Oath or Declaration [Total Pages]
   a. [ ] Newly executed (original or copy)
   b. [ ] Copy from a prior application (37 C.F.R. § 1.63(d))
   (for continuation/divisional with Box 16 completed)
      i. [ ] DELETION OF INVENTOR(S)
      Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. §§ 1.63(d)(2) and 1.33(b).

NOTE FOR ITEMS 1 & 13: IN ORDER TO BE ENTITLED TO PAY SMALL ENTITY FEES, A SMALL ENTITY STATEMENT IS REQUIRED (37 C.F.R. § 1.27), EXCEPT IF ONE FILED IN A PRIOR APPLICATION IS RELIED UPON (37 C.F.R. § 1.28).

5. [ ] Microfiche Computer Program *(Appendix)*

6. Nucleotide and/or Amino Acid Sequence Submission
(if applicable, all necessary)
   a. [ ] Computer Readable Copy
   b. [ ] Paper Copy (identical to computer copy)
   c. [ ] Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

7. [ ] Assignment Papers (cover sheet & document(s))
8. [ ] 37 C F.R.§3.73(b) Statement *(when there is an assignee)* [ ] Power of Attorney
9. [ ] English Translation Document *(if applicable)*
10. [ ] Information Disclosure Statement (IDS)/PTO-1449 [ ] Copies of IDS Citations
11. [ ] Preliminary Amendment
12. [X] Return Receipt Postcard (MPEP 503)
(Should be specifically itemized)
13. [ ] * Small Entity Statement(s) *(PTO/SB/09-12)* [ ] Statement filed in prior application, Status still proper and desired
14. [ ] Certified Copy of Priority Document(s)
(if foreign priority is claimed)
15. [X] Other: certificate of mailing

16. **If a CONTINUING APPLICATION,** *check appropriate box, and supply the requisite information below and in a preliminary amendment:*
[ ] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No: _____/_____
Prior application information: *Examiner* _____ *Group / Art Unit:* _____
For CONTINUATION or DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 4b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

17. CORRESPONDENCE ADDRESS

[X] *Customer Number or Bar Code Label* **22883** *(Insert Customer No. or Attach bar code label here)* or [ ] *Correspondence address below*

PATENT TRADEMARK OFFICE

Name					
Address					
City		State		Zip Code	
Country		Telephone		Fax	

Name (Print/Type)	Steven A. Swernofsky	Registration No. (Attorney/Agent)	33,040
Signature	Steven A. Swernofsky	Date	August 18, 2000

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

103.1035.01

Certificate of Mailing (37 C.F.R. § 1.10)

I hereby certify that this paper (along with any paper referred to as being attached or enclosed) is being deposited with the United States Postal Services on the date shown below as "Express Mail" (Post Office to Addressee) in an envelope addressed to the Honorable Assistant Commissioner for Patents and Trademarks, Box Patent Application, Washington, D.C. 20231.

Mailing Label No. **EL 524 780 239 US**

Date of Deposit: August 18, 2000

Arlette Malhas
Printed Name

Arlette Malhas
Signature

Documents enclosed:

- Provisional Patent Application Transmittal Form; and Fee Transmittal Form with check for fees;
- Specification (26) pages;
- Claims (6) pages;
- Abstract (4) pages;
- Drawings (3) pages;
- Return post card; and
- Certificate of Express Mailing.

This application is submitted in the name of the following inventor(s):

Inventor	Citizenship	Residence City and State
Blake LEWIS	USA	Palo Alto, California
John EDWARDS	USA	Sunnyvale, California
Srinivasan VISWANATHAN	India	Fremont, California

The assignee is Network Appliance, Inc., a California corporation having an office at 495 East Java Drive, Sunnyvale, CA 94089.

Title of the Invention

Instant Snapshot

Background of the Invention

1. Field of Invention

This invention relates to data storage systems.

This invention relates to data storage systems.

2. Related Art

Snapshots of a file system capture the contents of the files and directories in a file system at a particular point in time. Such snapshots have several uses. They allow the users of the file system to recover earlier versions of a file following an unintended deletion or modification. The contents of the snapshot can be copied to tape to provide a backup copy of the file system, and it can be copied to another file server and used as a replica. File systems, including the WAFL (Write Anywhere File Layout) file system, include a copy-on-write snapshot mechanism. Snapshot block ownership is recorded by updating the block's entry in the block map file, a bitmap associated with the vacancy of blocks.

One problem with the prior art of creating snapshots is that the requirement for additional file system metadata in the active filesystem to keep track of which blocks snapshots occupy. This metadata requires 4 bytes per 4–KB file system block, i.e., $1/1024^{th}$ of the file system. These methods are inefficient both in their use of storage

space and in the time needed to create the snapshots.

A second problem with earlier snapshot implementations, was the time consuming steps of writing out a description of the snapshot state on creation and removing it on deletion.

A third problem with earlier copy-on-write mechanisms, was the required steps consumed a considerable amount of time and file system space. For example, some systems, such as those supplied with DCE/DFS include a copy-on-write mechanism for creating snapshots (called "clones"). The copy-on-write mechanism was used to record which blocks each clone occupied. Such systems require a new copy of the inode file and the indirect blocks for all files and directories are created when updating all of the original inodes.

Accordingly, it would be advantageous to provide an improved technique for more quickly and efficiently capturing the contents of the files and directories in the file system at a particular point in time. This is achieved in an embodiment of the invention that is not subject to the drawbacks of the related art.

Summary of the Invention

The invention provides an improved method and apparatus for creating a snapshot of a file system.

In a first aspect of the invention, a "copy-on-write" mechanism is used. An effective snapshot mechanism must be efficient both in its use of storage space and in the time needed to create it because file systems are often large. The snapshot uses the same blocks as the active file system until the active file system is modified. Whenever a modification occurs, the modified data is copied to a new block and the old data is saved (henceforth called "copy-on-write"). In this way, the snapshot only uses space where it differs from the active file system, and the amount of work required to create the snapshot initially is small.

In a second aspect of the invention, a record of which blocks are being used by the snapshot is included in the snapshot itself, allowing effectively instantaneous snapshot creation and deletion.

In a third aspect of the invention, the state of the active file system is described by a set of metafiles; in particular, a bitmap (henceforth the "active map") describes which blocks are free and which are in use by the active file system. The inode file describes which blocks are used by each file, including the metafiles. The inode file itself is described by a special root inode, also known as the "fsinfo block. This copy of the root inode becomes the root of the snapshot. The root inode captures all required states for creating the snapshot such as the location of all files and directories in the file system, it. During subsequent updates of the active file system, the system consults the bitmap included in the snapshot (the "snapmap") to determine whether a block is free for reuse or belongs to a snapshot. This mechanism allows the active file system to keep track of which blocks each snapshot uses without recording any additional bookkeeping information in the file system.

In a fourth aspect of the invention, a snapshot can also be deleted instantaneously simply by discarding its root inode. Further bookkeeping is not required, because the snapshot includes it's own description.

In a fifth aspect of the invention, the performance overhead associated with

the search for free blocks is reduced by the inclusion of a summary file. The summary file identifies blocks that are used by at least one snapshot; it is the logical OR of all the snapmap files. The write allocation code decides whether a block is free by examining the active map and the summary file. The active map indicates whether the block is currently in use in the active file system. The summary file indicates whether the block is used by any snapshot.

In a sixth aspect of the invention, the summary file is updated in the background after the creation or deletion of a snapshot. This occurs concurrently with other file system operations. Two bits are stored in the file system "fsinfo block" for each snapshot. These two bits indicate whether the summary file needs to be updated using the snapshot's snapmap information as a consequence of its creation or deletion. When a block is freed in the active file system, the corresponding block of the summary file is updated with the snapmap from the most recently created snapshot, if this has not already been done. An in-core bit map records the completed updates to avoid repeating them unnecessarily. This ensures that the combination of the active bitmap and the summary file will consistently identify all blocks that are currently in use. Additionally, the summary file is updated to reflect the effect of any recent snapshot deletions when

freeing a block in the active file system. This allows reuse of blocks that are now entirely free. After updating the summary file following a snapshot creation or deletion, the corresponding bit in the fsinfo block is adjusted.

In a seventh aspect of the invention, the algorithm for deleting a snapshot involves examining the snapmaps of the deleted snapshot and the snapmaps of the next oldest and next youngest snapshot. A block that was used by the deleted snapshot but is not used by its neighbors can be marked free in the summary file, as no remaining snapshot is using it. However, these freed blocks cannot be reused immediately, as the snapmap of the deleted snapshot must be preserved until summary updating is complete. During a snapdelete free blocks are found by using the logical OR of the active bitmap, the summary file, and the snapmaps of all snapshots for which post-deletion updating is in progress. In other words, the snapmap of the deleted snapshot protects the snapshot from reuse until it is no longer needed for updating.

In the preferred embodiment, the invention is operative on WAFL file system. However, it is still possible for the invention to be applied to any computer data storage system such as a database system or a store and forward system such as cache or

RAM if the data is kept for a limited period of time.

Brief Description of the Drawings

Figure 1 shows a block diagram of a system for an instant snapshot.

Figure 2 shows a block diagram of an instant snapshot.

Figure 3 shows a flow diagram of a method for creating a snapshot.

Incorporated Disclosures

The inventions described herein can be used in conjunction with inventions described in the following applications:

- U.S. Patent Application Serial No. _____, Express Mail Mailing No. EL 524781089US, filed August 18, 2000, in the name of Blake LEWIS, attorney docket number 103.1033.01, titled "Reserving File System Blocks"

- U.S. Patent Application Serial No. _____, Express Mail Mailing No. EL524780242US, filed August 18, 2000, in the name of Rajesh SUNDARAM, attorney docket number 103.1034.01, titled "Dynamic Data Storage"

- U.S. Patent Application Serial No. _____, Express Mail Mailing No. EL524780256US, filed August 18, 2000, in the name of Ray CHEN, attorney docket number 103.1047.01, titled "manipulation of Zombie Files and Evil-Twin Files"

- Application Serial Number _____, in the names of Scott SCHOENTHAL, Express Mailing Number EL524781075US, titled "Persistent and Reliable Delivery of Event Messages", assigned to the same assignee, attorney docket number 103.1048.01, and all pending cases claiming the priority thereof.

and

- Application Serial Number _____, in the names of Douglas P. DOUCETTE, Express Mailing Number EL524781092US, titled "Improved Space Allocation in a Write Anywhere File System", assigned to the same assignee, attorney docket number 103.1045.01, and all pending cases claiming the priority thereof.

Detailed Description of the Preferred Embodiment

In the following description, a preferred embodiment of the invention is described with regard to preferred process steps and data structures. However, those skilled in the art would recognize, after perusal of this application, that embodiments of the invention might be implemented using a variety of other techniques without undue experimentation or further invention, and that such other techniques would be within the scope and spirit of the invention.

Lexicography

As used herein, use of the following terms refer or relate to aspects of the invention as described below. The general meaning of these terms is intended to be illustory and in no way limiting.

- **Inode** - In general, the term "inode" refers to data structures that include information about files in Unix and other file systems. Each file has an inode and is identified by an inode number (i-number) in the file system where it resides. Inodes provide

important information on files such as user and group ownership, access mode (read, write, execute permissions) and type. An inode points to the file blocks or indirect blocks of the file it represents.

- **Sector** - In general, the term "sector" refers to a physical section of a disk drive including a collection of bytes, such as 512 bytes.

- **Data Storage Block** - In general, the phrase "data storage block" refers to specific allocation areas on a hard disk. The allocation area is a collection of sectors, such as 8 sectors or 4,096 bytes, commonly called 4K bytes or 4-KB.

- **File Block** - In general, the phrase "file block" refers to a standard size block of data including some or all of the data in a file. In the preferred embodiment, the file block is the same size as a data storage block.

- **fsinfo (File System Information Block)** - In general, the phrase "file system information block" refers to one or more copies of a block known as the "fsinfo block". These blocks are located at fixed locations on the disks. The fsinfo block

includes data about the volume including the size of the volume, volume level options, language and more.

- **WAFL (Write Anywhere File Layout)** - In general, the term "WAFL" refers to a high level structure for a file system. Pointers are used for locating data. All the data is included in files. These files can be written anywhere on the disk in chunks of file blocks placed in data storage blocks.

- **Volume** In general, the term "volume" refers to a single file system. The file system may be composed of a collection of disk drives.

- **Consistency Point (CP)** - In general, the term "CP" refers to a time that a file system reaches a consistent state. When this state is reached, all the files have been written to all the blocks and are safely on disk and the one or more copies of redundant fsinfo blocks get written out. If the system crashes before the fsinfo blocks go out, all other changes are lost and the system reverts back to the last CP. The file system advances atomically from one CP to the next.

- **Consistent State** - In general, the phrase "consistent state" refers to the system configuration of files in blocks after the CP is reached.

- **Range** - In general, the term "range" refers to a group of blocks, such as 1,024 blocks.

- **Active file system** - In general, the phrase "active file system" refers to the current file system arrived at with the most recent CP. In the preferred embodiment, the active file system includes the active map, the summary map and points to all snapshots and other data storage blocks through a hierarchy of inodes, indirect data storage blocks and more.

- **Active map** - In general, the phrase "active map" refers to a to a file including a bitmap associated with the vacancy of blocks of the active file system.

- **Snapshot** - In general, the term "snapshot" refers to a copy of the file system. The snapshot diverges from the active file system over time as the active file system is modified. A snapshot can be used to return the file system to a particular CP (consistency point).

- **Snapmap** - In general, the term "snapmap" refers to a file including a bitmap associated with the vacancy of blocks of a snapshot. The active map diverges from a snapmap over time as the blocks used by the active file system change during consistency points.

- **Summary map** - In general, the term "summary map" refers to a file including an IOR (inclusive OR) bitmap of all the snapmaps.

- **Space map** - In general, the term "space map" refers to a file including an array of numbers which describe the number of storage blocks used in an allocation area.

- **Blockmap** - In general, the term "blockmap" refers to a map describing the status of the blocks in the file system.

- **Snapdelete** - In general, the term "snapdelete" refers to an operation that removes a particular snapshot from the file system. This command can allow a storage block to be freed for reallocation provided no other snapshot or the active file system uses the storage block.

- **Snapcreate** – In general, the term "snapcreate" refers to the operation of retaining a consistency point and preserving it as a snapshot.

As described herein, the scope and spirit of the invention is not limited to any of the definitions or specific examples shown therein, but is intended to include the most general concepts embodied by these and other terms.

System Elements

Figure 1 shows a block diagram of a system for an instant snapshot.

The root block 100 includes the inode of the inode file 105 plus other information regarding the active file system 110, the active map 115, previous active file systems known as snapshots 120, 125, 130 and 135 and their respective snapmaps 140, 145, 150 and 155.

The active map 115 of the active file system 110 is a bitmap associated with the vacancy of blocks for the active file system 110. The respective snapmaps 140,

145, 150 and 155 are active maps can be associated with particular snapshots 120, 125, 130 and 135 and an inclusive OR summary map 160 of the snapmaps 140, 145, 150 and 155. Also shown are other blocks 115 including double indirect blocks 130 and 132, indirect blocks 165, 166 and 167 and data blocks 170, 171, 172 and 173. Finally, Figure 1 shows the spacemap 180 including a collection of spacemap blocks of numbers 181, 182, 183, 184 and 190.

The root block 100 includes a collection of pointers that are written to the file system when the system has reached a new CP (consistency point). The pointers are aimed at a set of indirect (or triple indirect, or double indirect) inode blocks (not shown) or directly to the inode file 105 consisting of a set of blocks known as inode blocks 191, 192, 193, 194 and 195.

The number of total blocks determines the number of indirect layers of blocks in the file system. The root block 100 includes a standard quantity of data, such as 128 bytes. 64 of these 128 bytes describe file size and other properties; the remaining 64 bytes are a collection of pointers to the inode blocks 191, 192, 193, 194 and 195 in the inode file 105. Each pointer in the preferred embodiment is made of 4 bytes. Thus, there

are approximately 16 pointer entries in the root block 100 aimed at 16 corresponding inode blocks of the inode file 105 each including 4K bytes. If there are more than 16 inode blocks, indirect inode blocks are used.

In a preferred embodiment, file blocks are 4096 bytes and inodes are 128 bytes. It follows that each block of the inode file contains 32 (i.e. 4,096/128) separate inodes that point to other blocks 115 in the active file system.

Inode block 193 in the inode file 105 points to a set of blocks (1, 2, 3, ..., P) called the active map 115. Each block in the active map 115 is a bitmap where each bit corresponds to a block in the entire volume. A "1" in a particular position in the bitmap correlates with a particular allocated block in the active file system 110. Conversely, a "0" correlates to the particular block being unused by the active file system 110. Since each block in the active map 115 can describe up to 32K blocks or 128 MB, 8 blocks are required per GB, 8K blocks per TB.

Another inode block in the inode file 105 is inode block N 212. This block includes a set of pointers to a collection of snapshots 120, 125, 130 and 135 of the

volume. Each snapshot includes all the information of a root block and is equivalent to an older root block from a previous active file system. The snapshot 120 may be created at any past CP. Regardless when the snapshot is created, the snapshot is an exact copy of the active file system at that time. The newest snapshot 120 includes a collection of pointers that are aimed directly or indirectly to the same inode file 105 as the root block 100 of the active file system 110. As the active file system 110 changes (generally from writing files, deleting files, changing attributes of files, renaming file, modifying their contents and related activities), the active file system and snapshot will diverge over time. Given the slow rate of divergence of an active file system from a snapshot, any two snapshots will share many of the same blocks. The newest snapshot 120 is associated with snapmap 140. Snapmap 140 is a bit map that is initially identical to the active map 115. The older snapshots 125, 130 and 194 have a corresponding collection of snapmaps 145, 150 and 155. Like the active map 115, these snapmaps 145, 150 and 155 include a set of blocks including bitmaps that correspond to allocated and free blocks for the particular CP when the particular snapmaps 145, 150 and 155 were created. Any active file system may have a structure that includes pointers to one or more snapshots. Snapshots are identical to the active file system when they are created. It follows that snapshots contain pointers to older snapshots. There can be a large number of previous

snapshots in any active file system or snapshot. In the event that there are no snapshot, there will be no pointers in the active file system.

Blocks not used in the active file system 110 are not necessarily available for allocation or reallocation because the blocks may be used by snapshots. Blocks used by snapshots are freed by removing a snapshot using the snapdelete command. When a snapshot is deleted any block used only by that snapshot and not by other snapshots nor by the active file system becomes free for reuse by WAFL. If no other snapshot or active files uses the block, then the block can be freed and written over during the next copy on-wrote-execution by WAFL. The system can relatively efficiently determine whether a block can be removed using the "nearest neighbor rule". If the previous and next snapshot do not allocate a particular block in their respective snapmaps, then the block can be freed for reuse by WAFL. For WAFL to find free space to write new data or metadata, it could search the active map 115 and the snapmaps (140, 145, 150 and 155) of the snapshots (120, 125, 130 and 135) to find blocks that are totally unused. This would be very inefficient; thus it is preferable to use the active map and the summary map as described below

A summary map 160 is created by using an IOR (inclusive OR) operation 139 on the snapmaps 140, 145, 150 and 155. Like the active map 115 and the snapmaps 140, 145, 150 and 155, the summary map 160 is a file whose data blocks (1, 2, 3, ...Q) contained a bit map. Each bit in each block of the summary map. describes the allocation status of one block in the system with "1" being allocated and "0" being free. The summary map 160 describes the allocated and free blocks of the entire volume from all the snapshots 120, 125, 130 and 135 combined. The use of the summary file 160 is to avoid overwriting blocks in use by snapshots.

An IOR operation on sets of blocks (such as 1,024 blocks) of the active map 115 and the summary map 160 produces a spacemap 180. Unlike the active map 115 and the summary map 160, which are a set of blocks containing bitmaps, the spacemap 180 is a set of blocks including 181, 182, 183, 184, and 190 containing arrays of binary numbers. The binary numbers in the array represent the addition of all the vacant blocks in a region containing a fixed number of blocks, such as 1,024 blocks. The array of binary numbers in the single spacemap block 181 represents the allocation of all blocks for all snapshots and the active file system in one range of 1,024 blocks. Each of the binary numbers 181, 182, 183, 184, and 190 in the array are a fixed length. In a

preferred embodiment, the binary numbers are 16 bit numbers, although only 10 bits are used.

In a preferred embodiment, the large spacemap array binary number 182 (0000001111111110=1,021 in decimal units) tells the file system that the corresponding range is relatively full. In such embodiments, the largest binary number 00001111111111 (1,023 in decimal) represents a range containing at most one empty.. The small binary number 184 (0000000000001110=13 in decimal units) instructs the file system that the related range is relatively empty. The spacemap 180 is thus a representation in a very compact form of the allocation of all the blocks in the volume broken into 1,024 block sections. Each 16 bit number in the array of the spacemap 180 corresponds to the allocations of blocks in the range containing 1,024 blocks or about 4 MB. Each spacemap block 180 has about 2,000 binary numbers in the array and they describe the allocation status for 8 GB. Unlike the summary map 120, the spacemap block 180 needs to be determined whenever a file needs to be written.

Figure 2 shows a block diagram of an instant snapshot.

The old root block 200 of snapshot #1 201 includes the inode of the inode file 202 plus other information regarding the previous active file system known as snapshot #1 201, the snap map 205, earlier active file systems known as snapshot #2 210, snapshot #3 215 and snapshot #4 220, and their respective snapmaps 225, 230 and 235.

The snapmap 205 of the previous active file system, snapshot #1 201, is a bitmap associated with the vacancy of blocks for snapshot #1 201. The respective snapmaps 225, 230 and 235 are earlier active maps that can be associated with particular snapshots 210, 215 and 220 and an inclusive OR summary map 245 of the snapmaps 225, 230 and 235. Also shown are other blocks 211 including double indirect blocks 240 and 332, indirect blocks 250, 251 and 252 and data blocks 260, 262, 263 and 264. Finally, Figure 2 shows the spacemap 270 of snapshot #1 201 including a collection of spacemap blocks of binary numbers 272, 273, 274, 275 and 276.

The old root block 200 includes a collection of pointers that are written to the previous active file system when the system had reached the previous CP. The pointers are aimed at a set of indirect (or triple indirect, or double indirect) inode blocks (not shown) or directly to the inode file 202 consisting of a set of blocks known as inode

blocks 281, 282, 283, 284 and 285.

An inode block 281 in the inode file 202 points to other blocks 328 in the old root block 201 starting with double indirect blocks 240 and 332 (there could also be triple indirect blocks). The double indirect blocks 240 and 332 include pointers to indirect blocks 250, 251 and 252. The indirect blocks 250, 251 and 252 include pointers that are directed to data leaf blocks 260, 262, 263 and 264 of the active file system 201.

Inode block 283 in the inode file 202 points to a set of blocks (1, 2, 3, ..., P) called the snap map 205. Each block in the snap map 205 is a bitmap where each bit corresponds to a block in the entire volume. A "1" in a particular position in the bitmap correlates with a particular allocated block in the active file system 201. Conversely, a "0" correlates to the particular block being free for allocation in the old root block 201. Each block in the snap map 205 can describe up to 32K blocks or 128 MB.

Inode file 202 also includes inode block N 285. This block includes a set of pointers to a collection of earlier snapshots, snapshot #2 210, shapshot #3 215 and snapshot #4 220 of the volume. Each snapshot includes all the information of a root

block and is equivalent to an older root block from a previous active file system.

Snapshot #1 201 also includes an old summary map 245 and old spacemap blocks 270. Although these blocks of data are included in snapshot #1 201 and previous snapshots, in a preferred embodiment, this data is not used by the active file system of figure 2.

Method of Use

Figure 3 shows a flow diagram of a method for using a system as shown in figure 1.

A method 300 is performed by the file system 100. Although the method 400 is described serially, the steps of the method 300 can be performed by separate elements in conjunction or in parallel, whether asynchronously, in a pipelined manner, or otherwise. There is no particular requirement that the method 300 be performed in the same order in which this description lists the steps, except where so indicated.

At a flow point 305, the file system 100 is ready to perform a method 300.

At a step 310, a user will request a snapshot of the file system 100.

At a step 315, a timer associated with the file system 100 initiates the creation of a new snapshot.

At a step 320, the file system 100 receives a request to make a snapshot.

At a step 325, the file system 100 creates a new file.

At a step 330, the root node of the new file points to the root node of the current active file system.

At a step 335, the file system 100 makes the file read only.

At a step 340, the file system 100 updates the new summary map by using an inclusive OR of the most recent snapmap and the existing summary file. This step

must be done before any blocks are freed in the corresponding active map block. If multiple snapshots are created such that the processing overlaps in time, the update in step 340 need only be done for the most recently created snapshot.

At a flow point 345, the snapshot create and the summary file update is completed and the snapshot creation is done.

An analogous method may be performed for snapshot delete.

Alternative Embodiments

Although preferred embodiments are disclosed herein, many variations are possible which remain within the concept, scope, and spirit of the invention, and these variations would become clear to those skilled in the art after perusal of this application.

Claims

1 A method for capturing the contents of the files and directories in a file system, said file system comprising a set of storage blocks in a mass storage system including steps for

recording an active map in said file system of said storage blocks not available for writing data;

recording a consistency point in said file system including a consistent version of said file system at a previous time, said consistency point including a copy of said active map at said previous time;

refraining from writing data to storage blocks in response to said active map; and

at least one of said copy of said active map at said previous time.

2. A method as in claim 1, wherein said step for refraining includes determining a logical union of said storage blocks used by one or more of said copies of said active map at said previous time.

3. A method as in claim 1, wherein said step for refraining includes determining a subset of said storage blocks used by one or more of said copies of said active map at said previous time.

4. A method as in claim 1, wherein said file system is a WAFL file system.

5. A method as in claim 1, wherein said active map at said previous time is a snapmap.

6. A method as in claim 1 and 5, including removing a root inode of said snapmap using a snap delete.

7 A method as in claim 6, including steps for determining not to write to a block after said step, provided the previous or next snapmap uses said block.

8. A method as in claim 1, including a copy-on-write mechanism for copying modified data to a new block and saving old data in a current data block.

9. A method for capturing the contents of the files and directories in a file system, said file system comprising a set of storage blocks in a mass storage system including

recording a consistency point in said file system including a consistent version of said file system at a previous time, said consistency point including a copy of said active map at said previous time; and

returning to said file system at a previous time using said consistent version of said file system following an unintended deletion or modification.

10. A method as in claim 9, wherein said consistent version includes a pointer to a previous root block of the inode file.

11. A method as in claim 9, wherein said file system is a WAFL file system.

12. A method as in claim 9, wherein said active map at said previous time is a snapmap.

13. A method as in claim 9 and 12, including a snapdelete method for removing a root inode of said snapmap.

14. A method as in claim 13, including steps for determining not to write to a block after said snapdelete method provided a previous or next snapmap uses said block.

15. A method as in claim 9, including a copy-on-write mechanism for copying modified data to a new block and saving old data in a current data block.

16. A method for saving previous versions of an active file system including the contents of the files and directories in a file system, said file system comprising a set of storage blocks in a mass storage system including steps for

writing modified files to unused data blocks;

keeping previous files in currently occupied blocks; and

recording a consistency point in said file system including a consistent version of said file system at a previous time, said consistency point including a copy of said active map at said previous time;

17. A method as in claim 16, including retrieving said file system at a previous time using a pointer.

18. A method as in claim 16, wherein said pointer corresponds to a root block of said file system at a previous time.

19. A method as in claim 16, wherein said file system is a WAFL file system.

20. A method as in claim 16, wherein said active map at said previous time is a snapmap.

21. A method as in claim 16 and 20, including a snapdelete method for

removing a root inode of said snapmap.

22. A method as in claim 20, including not writing to a block after said snapdelete method provided a previous or next snapmap uses said block.

23. A method as in claim 16, including a copy-on-write mechanism for copying modified data to a new block and saving old data in a current data block.

Abstract of the Disclosure

The invention provides an improved method and apparatus for creating a snapshot of a file system. In a first aspect of the invention, a "copy-on-write" mechanism is used. An effective snapshot mechanism must be efficient both in its use of storage space and in the time needed to create it because file systems are often large. The snapshot uses the same blocks as the active file system until the active file system is modified. Whenever a modification occurs, the modified data is copied to a new block and the old data is saved (henceforth called "copy-on-write"). In this way, the snapshot only uses space where it differs from the active file system, and the amount of work required to create the snapshot is small. In a second aspect of the invention, a record of which blocks are being used by the snapshot is included in the snapshot itself, allowing effectively instantaneous snapshot creation and deletion. In a third aspect of the invention, the state of the active file system is described by a set of metafiles; in particular, a bitmap (henceforth the "active map") describes which blocks are free and which are in use. The inode file describes which blocks are used by each file, including the metafiles. The inode file itself is described by a special root inode, also known as the "fsinfo block". The system begins creating a new snapshot by making a copy of the root

inode. This copy of the root inode becomes the root of the snapshot. The root inode captures all required states for creating the snapshot such as the location of all files and directories in the file system, it. During subsequent updates of the active file system, the system consults the bitmap included in the snapshot (the "snapmap") to determine whether a block is free for reuse or belongs to the snapshot. This mechanism allows the active file system to keep track of which blocks each snapshot uses without recording any additional bookkeeping information in the file system. In a fourth aspect of the invention, a snapshot can also be deleted instantaneously simply by discarding its root inode. Further bookkeeping is not required, because the snapshot includes it's own description. In a fifth aspect of the invention, the performance overhead associated with the search for free blocks is reduced by the inclusion of a summary file. The summary file identifies blocks that are used by at least one snapshot; it is the logical OR of all the snapmap files. The write allocation code decides whether a block is free by examining the active map and the summary file. The active map indicates whether the block is currently in use in the active file system. The summary file indicates whether the block is used by any snapshot. In a sixth aspect of the invention, the summary file is updated in the background after the creation or deletion of a snapshot. This occurs concurrently with other file system operations. Two bits are stored in the file system "fsinfo block"

for each snapshot. These two bits indicate whether the summary file needs to be updated using the snapshot's snapmap information as a consequence of its creation or deletion. When a block is freed in the active file system, the corresponding block of the summary file is updated with the snapmap from the most recently created snapshot, if this has not already been done. An in-core bit map records the completed updates to avoid repeating them unnecessarily. This ensures that the combination of the active bitmap and the summary file will consistently identify all blocks that are currently in use. Additionally, the summary file is updated to reflect the effect of any recent snapshot deletions when freeing a block in the active file system. This allows reuse of blocks that are now entirely free. After updating the summary file following a snapshot creation or deletion, the corresponding bit in the fsinfo block is adjusted. In a seventh aspect of the invention, the algorithm for deleting a snapshot involves examining the snapmaps of the deleted snapshot and the snapmaps of the next oldest and next youngest snapshot. A block that was used by the deleted snapshot but is not used by its neighbors can be marked free in the summary file, as no remaining snapshot is using it. However, these freed blocks cannot be reused immediately, as the snapmap of the deleted snapshot must be preserved until summary updating is complete. During a snapdelete, free blocks are found by using the logical OR of the active bitmap, the summary file, and the snapmaps of all snapshots

for which post-deletion updating is in progress. In other words, the snapmap of the deleted snapshot protects the snapshot from reuse until it is no longer needed for updating. In the preferred embodiment, the invention is operative on WAFL file system. However, it is still possible for the invention to be applied to any computer data storage system such as a database system or a store and forward system such as cache or RAM if the data is kept for a limited period of time.

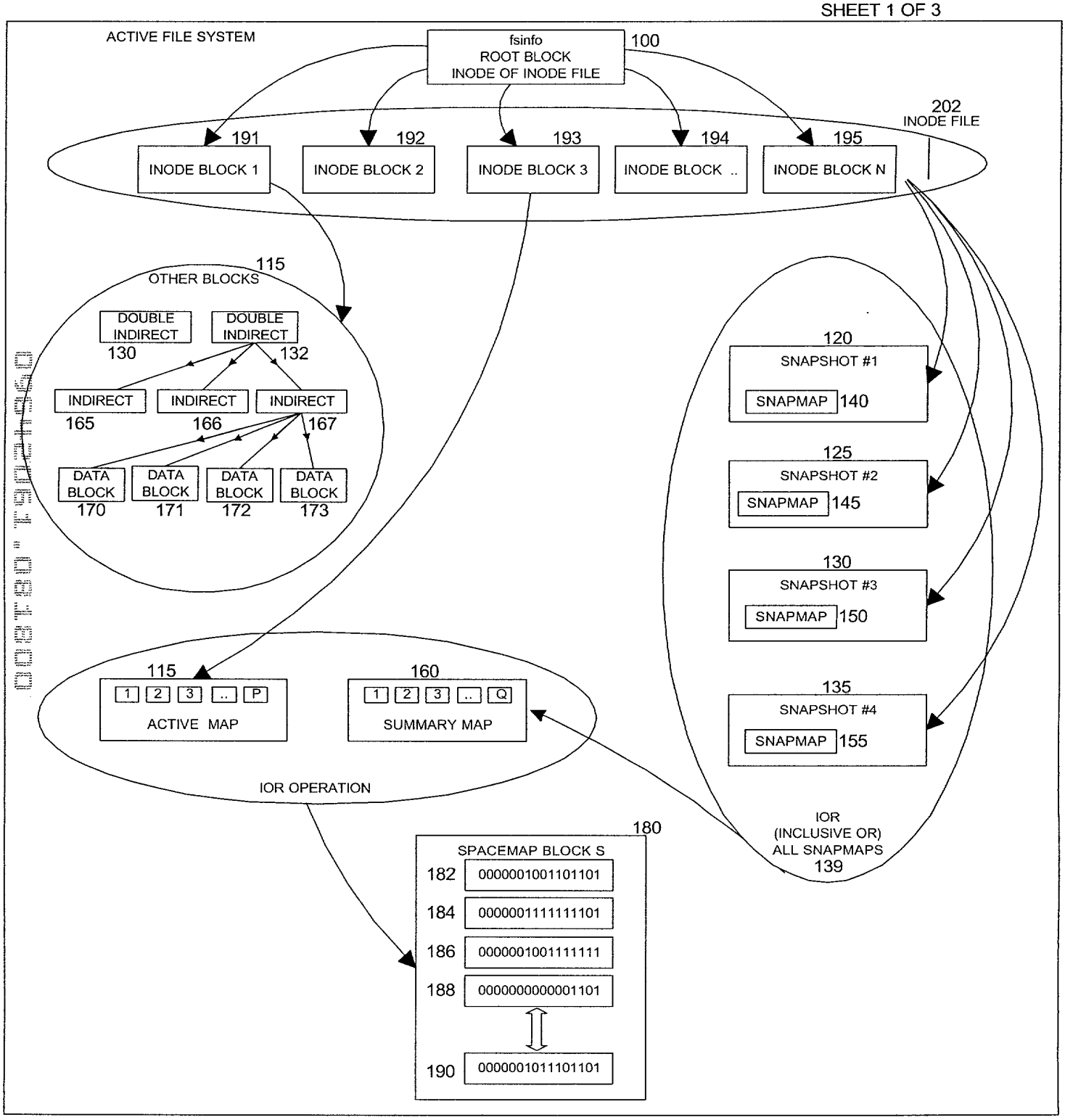
ACTIVE FILE SYSTEM
fsinfo
ROOT BLOCK
INODE OF INODE FILE
100
202
INODE FILE
191
192
193
194
195
INODE BLOCK 1
INODE BLOCK 2
INODE BLOCK 3
INODE BLOCK ..
INODE BLOCK N
115
OTHER BLOCKS
DOUBLE INDIRECT
DOUBLE INDIRECT
130
132
INDIRECT
INDIRECT
INDIRECT
165
166
167
DATA BLOCK
DATA BLOCK
DATA BLOCK
DATA BLOCK
170
171
172
173
120
SNAPSHOT #1
SNAPMAP 140
125
SNAPSHOT #2
SNAPMAP 145
130
SNAPSHOT #3
SNAPMAP 150
135
SNAPSHOT #4
SNAPMAP 155
115
1 2 3 .. P
ACTIVE MAP
160
1 2 3 .. Q
SUMMARY MAP
IOR OPERATION
IOR
(INCLUSIVE OR)
ALL SNAPMAPS
139
180
SPACEMAP BLOCK S
182 0000001001101101
184 0000001111111101
186 0000001001111111
188 0000000000001101
190 0000001011101101

FIG. 1

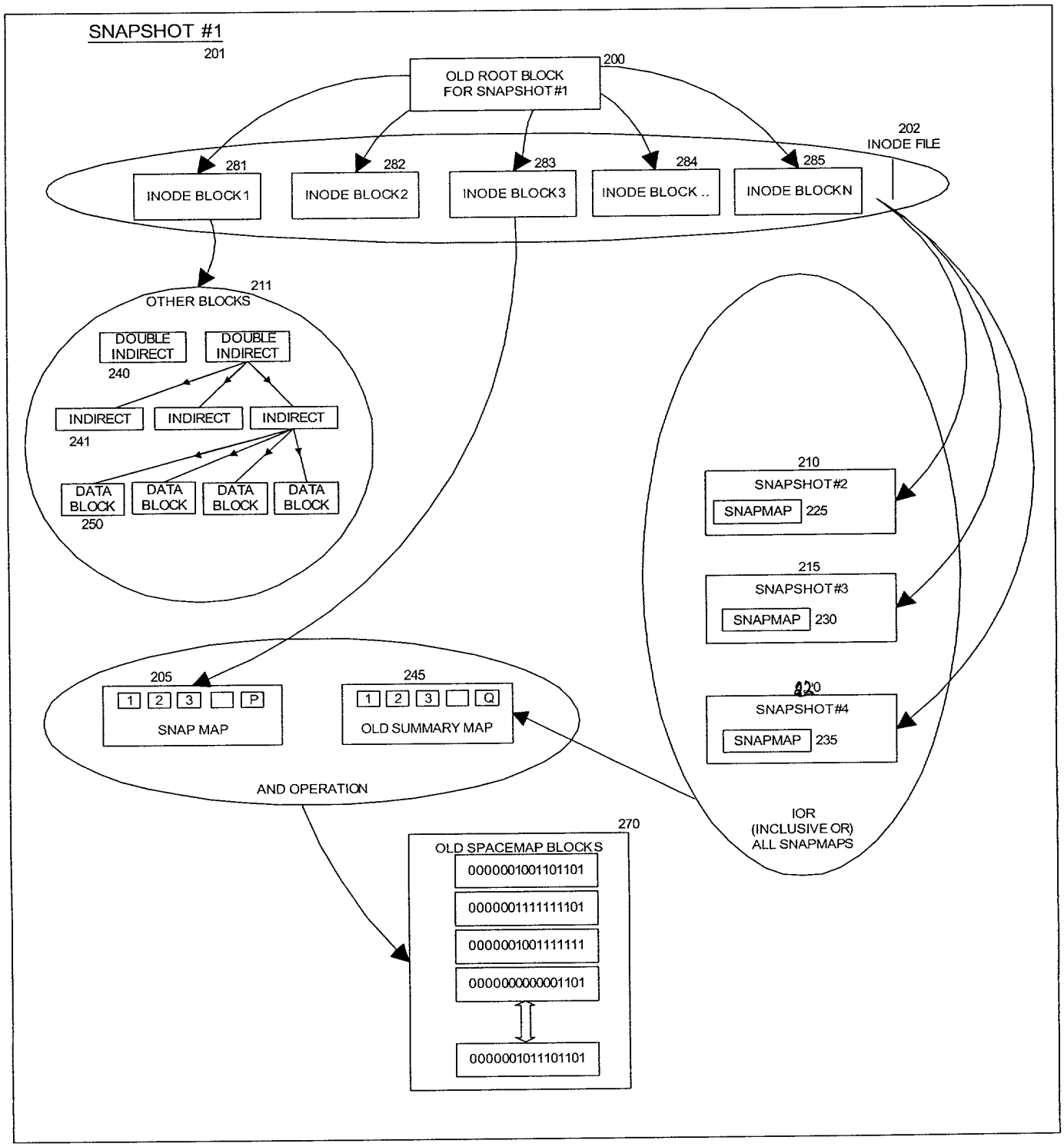
SNAPSHOT #1
201
OLD ROOT BLOCK FOR SNAPSHOT#1
200
202
INODE FILE
281
INODE BLOCK1
282
INODE BLOCK2
283
INODE BLOCK3
284
INODE BLOCK ..
285
INODE BLOCKN
211
OTHER BLOCKS
DOUBLE INDIRECT
DOUBLE INDIRECT
240
INDIRECT
INDIRECT
INDIRECT
241
DATA BLOCK
DATA BLOCK
DATA BLOCK
DATA BLOCK
250
210
SNAPSHOT#2
SNAPMAP 225
215
SNAPSHOT#3
SNAPMAP 230
220
SNAPSHOT#4
SNAPMAP 235
205
1 2 3 P
SNAP MAP
245
1 2 3 Q
OLD SUMMARY MAP
AND OPERATION
IOR
(INCLUSIVE OR)
ALL SNAPMAPS
270
OLD SPACEMAP BLOCKS
0000001001101101
0000001111111101
0000001001111111
0000000000001101
0000001011101101

FIG. 2

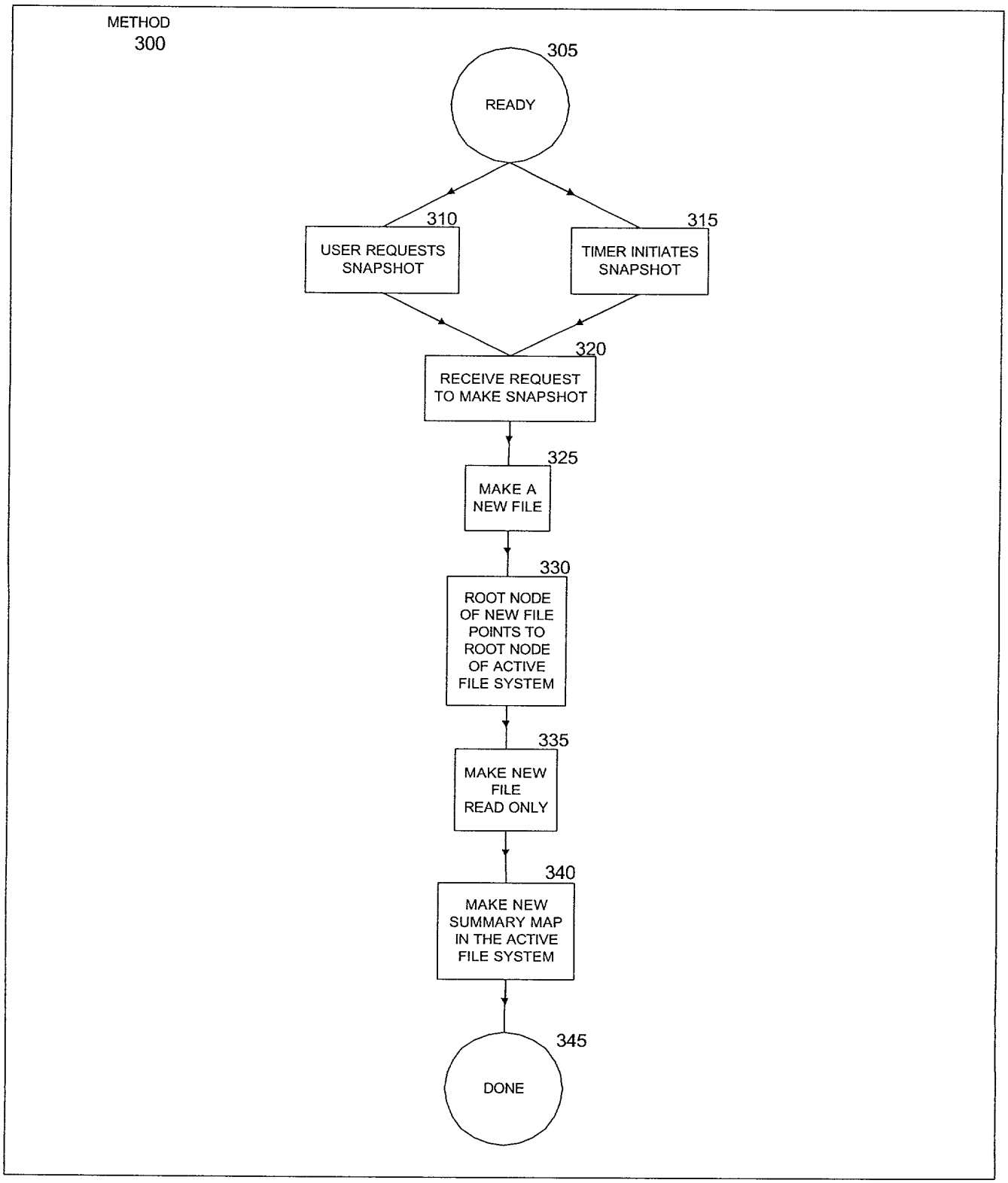
METHOD
300
305
READY
310
USER REQUESTS SNAPSHOT
315
TIMER INITIATES SNAPSHOT
320
RECEIVE REQUEST TO MAKE SNAPSHOT
325
MAKE A NEW FILE
330
ROOT NODE OF NEW FILE POINTS TO ROOT NODE OF ACTIVE FILE SYSTEM
335
MAKE NEW FILE READ ONLY
340
MAKE NEW SUMMARY MAP IN THE ACTIVE FILE SYSTEM
345
DONE

FIG. 3